SOR AND MGR[ν] EXPERIMENTS ON THE CRYSTAL MULTICOMPUTER

by

David Kamowitz

Computer Sciences Technical Report #623

January 1986

SOR and MGR[ν] Experiments on the Crystal Multicomputer*

by

David Kamowitz

ABSTRACT

This report describes distributed implementations of the red/black SOR algorithm and of the MGR[ν] multigrid algorithm on the Crystal multicomputer. Rates of convergence and observed efficiencies for both algorithms are compared.

*Supported by the Air Force Office of Scientific Research under Contract No. AFOSR-82-0275 and by NSF grant MCS-8105904.

1. Introduction

Various iterative methods, and multigrid in particular, are often used to solve the large linear systems that arise from the solution of elliptic partial differential equations. This report describes an experimental study undertaken to investigate the use of the Crystal distributed computing facility [1] to implement two of these methods. The two methods chosen for study are the red/black Successive Over-relaxation (SOR) method and the MGR[ν] multigrid algorithm.

Red/black SOR can be completely distributed to a number of machines and the algorithm itself is very easy to implement, hence it is a good test algorithm. However ρ, the rate of convergence (contraction number) for SOR is

$$\rho \sim 1 - ch$$

where c is some constant independent of h. This rate becomes abysmally slow as $h \to 0$. This slow rate suggests the use of much faster, although significantly more complicated, algorithms such as multigrid.

The MGR[ν] multigrid algorithm presents an added challenge beyond the details of the SOR algorithm. On the one hand multigrid algorithms have rates of convergence which are bounded away from one by constants independent of h. In fact, for Poisson's equation in a square the asymptotic rate of convergence $\rho(\nu)$ of the two grid MGR[ν] algorithm satisfies

$$\rho(\nu) \uparrow \frac{1}{2}\left(\frac{2\nu}{2\nu+1}\right)^{\frac{2\nu}{2\nu+1}} \text{ as } h \to 0 .$$

This is clearly superior to the rate of convergence for the SOR algorithm. On the other hand achieving this rate of convergence requires that much more work be performed. Indeed, from a distributed computing point of view there is a stage of the multigrid algorithm which is performed sequentially.

One can reasonably ask: Given the Crystal distributed computing facility can the completely distributable SOR algorithm be made faster than the MGR[ν] algorithm? Before answering this question decisions need to be made about the implementation of both of these algorithms.

This report has the following organization: Section two describes the specific differential equation and resulting linear system that was actually solved. Section three describes the Crystal multicomputer and the modifications to the sequential algorithms that are necessary for their implementation. Sections four and five

contain, respectively, details about the implementation and results for the SOR and for the MGR[ν] methods. Finally Section six contains some concluding remarks.

2. The Problem

The following analytic problem was chosen for testing by the two methods of solution.

$$\text{Find } u(x,y) \text{ such that}$$

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0 \text{ for } (x,y) \in \Omega, \tag{2.1}$$

$$u(x,y) = 0 \text{ for } (x,y) \in \partial\Omega$$

$$\text{and } \Omega \equiv \left\{ (x,y) \in R^2 \mid 0 < x,y < 1 \right\}.$$

Although the answer to this problem is obvious, $u = 0$, when studying iterative methods for the general Poisson equation with Dirichlet boundary conditions this homogeneous problem is the only problem which needs to be considered.

In general terms the method used to solve (2.1) is to discretize the region Ω into Ω_h and then to use either SOR or multigrid to solve the resulting linear system of equations.

The first step is to define Ω_h, the grid used to approximate Ω. Ω_h is constructed by first choosing N (N odd), the number of points on a side of Ω_h and setting h, the mesh width, equal to $\frac{1}{N+1}$. Then

$$\Omega_h \equiv \left\{ (x_i,y_j) \in \Omega \mid x_i = ih,\ y_j = jh,\ 1 \le i,j \le N \right\}.$$

For reasons that will become clearer later, Ω_h is divided into two subsets Ω_h^R and Ω_h^B (R corresponds to 'red' points and B corresponds to 'black' points). Ω_h^R and Ω_h^B are defined by

$$\Omega_h^R \equiv \left\{ (x_i,y_j) \in \Omega_h \mid i+j \equiv 1 \bmod 2 \right\}$$

and

$$\Omega_h^B \equiv \left\{ (x_i,y_j) \in \Omega_h \mid i+j \equiv 0 \bmod 2 \right\}.$$

The linear system that arises from discretizing (2.1) is, for all $(x_i,y_j) \in \Omega_h$

$$\frac{1}{h^2}\left\{ -U_{i-1,j} - U_{i,j-1} - U_{i,j+1} - U_{i+1,j} + 4U_{i,j} \right\} = 0\,. \tag{2.2}$$

with

$$U_{i,j} = 0 \text{ for } i = 0, N+1,\ j = 0, N+1.$$

The notation $U_{i,j}$ corresponds to $U(x_i, y_j)$, etc. Since the right hand side of equation (2.2) is zero the $\frac{1}{h^2}$ term is neglected. The system of linear equations (2.2) is written in matrix form as AU = 0, where U is the vector of unknowns corresponding to points in Ω_h. The solution to this system of equations, U, gives an approximation to $u(x, y)$, the solution of (2.1), which is an $O(h^2)$ approximation.

3. Brief overview of Crystal

The Crystal multicomputer is composed of a network of VAX 750 computers, each of which can communicate with each other. Communication between machines is accomplished by means of messages. This report does not describe the technical details of the message transfers; [1] and [2] describe these details.

However, from an algorithm design viewpoint a number of details are important. Each message can consist of up to 512 words (2048 bytes) of information. Once a machine sends a message it is free to proceed with new work. On the receiving end, the messages are placed in a buffer until they are read by the receiving machine. If no message has arrived when the receiving machine is ready for one then it must wait for a message to arrive. The ratio of transmission time of messages to computational time (multiplications etc.) is large, thus it is to an algorithm's advantage to do as many computations as possible between sending a message and waiting for a response. Otherwise the machine must busy wait for a message to arrive.

Connected to the Crystal multicomputer are a number of VAX 780 computers, referred to as hosts. The individual machines in the network of VAX 750 computers are referred to as node machines. To run a particular experiment the experimenter is able to communicate with the node machines from the host machines. For example, in this application the problem size and other parameters are sent to the node machines which then proceed to solve the problem, either by the red/black SOR algorithm or by the MGR[ν] algorithm. One particular node machine, for this application, is called the *master* node and the other nodes are called *slave* nodes. The *master* node communicates with the host machines and takes care of various bookkeeping tasks, such as timing the algorithm.

3.1. Implementation on Crystal

The basic idea used to solve the discrete problem (2.2) on Crystal is to decompose the region Ω_h into a number of smaller regions Ω_h^k and to assign each region Ω_h^k to a different processor k. The decomposition of Ω_h is accomplished by choosing the number of machines to use, M, and then splitting Ω_h into horizontal strips each of height 1/M. Thus, $\Omega_h = \Omega_h^1 \cup \Omega_h^2 \cup \cdots \cup \Omega_h^M$ where

$$\Omega_h^k \equiv \left\{ (x_i, y_j) \in \Omega_h \mid \frac{k-1}{M} \leq jh \leq \frac{k}{M} \right\} \tag{3.1}$$

Figure 1 shows, for N = 7 and M = 3, Ω_h^k, corresponding to machine k for $k = 1, 2, 3$.

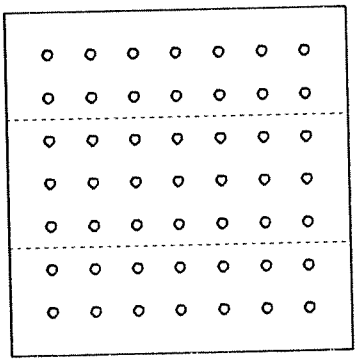


Figure 1 - Ω_h^k

Note that the number of points per processor does not have to be the same. In addition in our implementation an odd number of machines is required to insure that inter-machine boundaries do not lie on grid points.

3.2. Data Flow

One can apply the concept of data flow, see [3] for example, to study our particular implementation of each iteration of the red/black SOR algorithm on Crystal. The data flow concept involves examining what unknowns are actually being determined and what is the flow of data through the processors.

For our problem it is important to realize that the only true unknowns are those points lying on machine boundaries. Once these points are fixed then the points in the interior of Ω_h^k, for each iteration, are determined by the iteration process itself. In the eventual solution of (2.2) these interior points are determined by the uniqueness of the solution, which is implied by the maximum principle. Therefore the object of our parallel algorithm is to determine the values for these points. From this point of view the role of the points lying inside each Ω_h^k is solely to update these boundary values.

The task of each machine is thus to input boundary data and then to output updated values. Each machine proceeds independently and can be viewed as a single instruction—update boundary values. These computational processes are triggered solely by the flow of data through the network. There is no global synchronization or control over the algorithm with the exception of starting and stopping.

An important advantage in looking at the algorithm in this light is that the particular algorithm is independent of the architecture it is programmed in. For example the only change necessary in moving from the distributed Crystal network to a shared memory machine is that rather than physically transferring boundary values one would only have to mark the boundary rows as being free for the next machine to update.

Of course, for each machine the interior points serve as an initial guess for the next iteration. This has the practical implication that after the first iteration the machines are no longer interchangeable over Ω_h.

4. The SOR Method

The *successive over-relaxation*, SOR, method has been studied by many authors, see [4, 5, 6]. The red/black SOR method is characterized by its use of the decomposition of Ω_h into Ω_h^R and Ω_h^B. Although red/black line SOR (SLOR) also conveniently lends itself to the Crystal architecture, red/black point SOR is used instead since the decomposition of Ω_h into Ω_h^R and into Ω_h^B is also necessary for the MGR[ν] algorithm described in section five. This allows us to compare two methods for the solution of (2.2) based on the same basic iterative scheme. In [7] the red/black splitting is applied to vector processors.

The iterative scheme for our problem, for a given value of ω and initial guess U^0, is:

Repeat for each k, $k = 1, 2, 3, \cdots$ until convergence;

First update points in Ω_h^R:

For all $(x_i, y_j) \in \Omega_h^R$, set

$$U_{i,j}^{k+1} := \omega \left\{ U_{i-1,j}^k + U_{i,j-1}^k + U_{i,j+1}^k + U_{i+1,j}^k \right\} / 4 + (1-\omega) U_{i,j}^k .$$

Then update points in Ω_h^B:

For all $(x_i, y_j) \in \Omega_h^B$, set

$$U_{i,j}^{k+1} := \omega \left\{ U_{i-1,j}^k + U_{i,j-1}^k + U_{i,j+1}^k + U_{i+1,j}^k \right\} / 4 + (1-\omega) U_{i,j}^k .$$

Note that while updating points in Ω_h^R, $U_{i-1,j}^k$, $U_{i,j-1}^k$, $U_{i,j+1}^k$ and $U_{i+1,j}^k$ are all in Ω_h^B; similarly the reverse is true while updating points in Ω_h^B.

The algorithm is completely described by the above description except for the choice of ω. For $0 < \omega < 2$ the SOR algorithm converges [4]. In addition, for our problem the optimal ω is given by

$$\omega_{optimal} := \frac{2}{1 + \left(1 - \bar{\mu}^2\right)^{1/2}} ,$$

where $\bar{\mu}$ is the largest eigenvalue of the Jacobi iteration matrix. For general problems the value of $\bar{\mu}$ is not known and an iterative procedure must be used to determine $\omega_{optimal}$. However, in this case it is known that

$$\bar{\mu} = \cos \pi h.$$

In addition, for $\omega \equiv \omega_{optimal}$, the rate of convergence of the red/black SOR method is

$$\rho = \omega_{optimal} - 1 = \frac{1 - \sin\pi h}{1 + \sin\pi h} .$$

4.1. Implementation on Crystal

We have already seen that the basic idea used to exploit the Crystal architecture is the concept of domain decomposition. What is the interaction between machines in terms of the SOR iteration? What is the best reorganization of the algorithm to minimize the effects of the decomposition?

Each point $U_{i,j}$ that is updated using the SOR algorithm depends only upon its four nearest neighbors. For points interior to a given region Ω_h^k these four neighbors lie completely within Ω_h^k. However for boundary points of Ω_h^k (either the top or the bottom row) one of the neighbors lies in either Ω_h^{k-1} (points along the top of Ω_h^k) or in Ω_h^{k+1} (points along the bottom). Before each iteration machine k must therefore receive these boundary values from machines $k-1$ and $k+1$. Similarly machine k must send its boundary values to its two neighboring machines. Machines 1 and M have only one neighboring machine each, therefore there is only one boundary transfer for these two machines.

In order to "hide" the message transfers rather then sending boundary data and then waiting idly for the boundaries to arrive from neighboring machines, the interior points are first updated. Then, the points along the boundary of Ω_h^k are updated after receiving the boundary points from machines $k-1$ and $k+1$. In this way we hope that the time required to update the interior points will be larger than the time required for the boundaries to arrive; otherwise we must busy wait for them.

This procedure requires that rows in the original domain Ω_h that are boundaries of partitions Ω_h^k be represented twice. For example, the top row of Ω_h^k is updated by machine k and is also the bottom row, and hence used as boundary data, in machine $k-1$.

The decomposition of Ω_h into 'red' points and 'black' points allows machine k to work simultaneously with the other machines. Since the 'red' points depend upon values fixed in the 'black' points, we have already seen that all the 'red' points can be updated simultaneously. The only sharing of information is the values of 'black' points along machine boundaries. Similarly, to update the 'black' points, the 'red' points are held fixed, and again only the points along machine boundaries need to be exchanged.

In the case of lexicographical SOR, to update a given point, say $U_{i,j}$, points $U_{i-1,j}$ and $U_{i,j-1}$ must be updated first. In particular, if the point $U_{i,j}$ is in machine k, all of the points in machine $k-1$ must be first updated before updating $U_{i,j}$. Similarly the points in machines $k-2$, $k-3$, etc. must be updated before points in machine $k-1$. This loses the effect of distributing the computational work among a number of machines. Hence, a global decomposition of Ω_h, for example the red/black decomposition, is necessary for the Crystal implementation to succeed. In addition, since only points along machine boundaries are shared, one complete red/black cycle requires only two message transfers.

On Crystal the red/black SOR iteration for machine l, with N_l rows of unknowns, is:

Given ω and U^0:

Repeat for $k = 1, 2, 3, \cdots$ until convergence:

1. Send boundary data to neighbors.
2. Compute new U^{k+1} at interior points, rows $2, \cdots N_l - 1$.
3. Receive boundaries from machines $l-1$ and $l+1$.
4. Compute new U^{k+1} at boundary points of Ω_h^l.
5. If a complete red/black cycle has been performed, compute residual.

It is important to realize that the iterates computed by this distributed form of the red/black SOR algorithm are exactly the same as the iterates computed by the serial version of the red/black SOR algorithm. This allows us to easily compare the savings made by using the multicomputer.

A number of specific details about the programming of the algorithm are of interest. To control the various processors corresponding to each region Ω_h^k an additional processor is used. This additional processor collects the norms of the residuals from each machine k. The total norm over Ω_h is computed and sent to the host machine. In addition this machine provides each machine k with the necessary starting information (number of points, value of ω) and signals convergence to each machine k.

In order to keep the messages straight between machines, each message includes a synchronization number from its sender. In addition, when the messages arrive at their destinations the originator of each message is known. For example, machine k working on iteration l waits for messages sent from machines

$k-1$ and $k+1$ each labeled with synchronization number l. In the implementation of the red/black SOR algorithm this synchronization number corresponds to the iteration count of each machine. In case messages arrive too soon, for example from iteration $l+1$, then the messages are buffered until needed.

A final consideration concerning the implementation is worth noting. Although FORTRAN is available on Crystal, the language Wisconsin Modula is used instead. This choice is made to facilitate the buffering of the messages and because of Modula's superior choice of data structures. Since we are mainly interested in speedup and efficiency for this model implementation the consideration of which language to use, while important, does not change the conclusions based on the experimental results.

4.2. Experimental Results

The algorithm of the previous section was programmed and tested on the Crystal multicomputer. In order to compare the distributed version of the algorithm to the serial (single machine) version of the algorithm a number of definitions are required.

Let T_p be the time required to run the algorithm using p machines. Then the *speedup*, S_p, is

$$S_p \equiv \frac{T_1}{T_p}.$$

The *efficiency*, E_p, is

$$E_p \equiv \frac{S_p}{p}.$$

For this particular distribution of the work per iteration among the p machines in use, the minimum time required to run the algorithm is $\frac{T_1}{p}$, so

$$T_p \geq \frac{T_1}{p}.$$

Hence, E_p satisfies

$$E_p \equiv \frac{S_p}{p} = \frac{T_1}{pT_p} \leq 1.$$

One hopes for E_p to be as close to one as possible, however as we will see this is not always possible.

To measure the time required by the algorithm, the additional node which collected the norms from each piece timed the algorithm. By using a Crystal node machine to collect the times no allowances had to be

made for other users on the system. As an additional measure to insure accuracy three runs were made for every choice of N displayed in the tables. The deviation in time between runs was very small which supports the result that all of the measured time was due to the computations and not due to network traffic or other extraneous factors.

Runs were made with N equal to 15, 31, 63 and 127. This corresponds to 225, 961, 3969 and 16129 unknowns respectively. This may appear to be an unreasonable number of variables, however when using a larger or more complicated domain Ω these are realistic sized problems. In addition to varying N, the number of processors p equaled one, three, five, seven, nine and eleven.

Tables I through IV contain the results for N = 15, 31, 63 and 127 respectively. The column labeled maximum number of rows shows the number of rows in the largest division of Ω_h. Recall that the number of rows per machine is not required to be the same. The machine with the largest number of rows dominates the computation so it is important to compare the results for this machine.

Number of Machines	Maximum number of Rows	Average Time	Speedup	Efficiency
1	15	9.55	1.00	1.00
3	5	7.28	1.31	0.44
5	3	6.80	1.40	0.28
7	3	6.73	1.42	0.20
9	2	6.28	1.52	0.17
11	2	6.16	1.55	0.14

Table I - N = 15

Number of Machines	Maximum number of Rows	Average Time	Speedup	Efficiency
1	31	57.41	1.00	1.00
3	11	26.51	2.17	0.72
5	7	19.49	2.95	0.59
7	5	16.07	3.57	0.51
9	4	14.33	4.01	0.45
11	3	12.68	4.53	0.41

Table II - N = 31

Number of Machines	Maximum number of Rows	Average Time	Speedup	Efficiency
1	63	441.99	1.00	1.00
3	21	160.84	2.75	0.92
5	13	105.07	4.21	0.84
7	9	77.20	5.73	0.82
9	7	63.19	6.99	0.78
11	6	56.34	7.85	0.71

Table III - N = 63

Number of Machines	Maximum number of Rows	Average Time	Speedup	Efficiency
1	127	3675.56	1.00	1.00
3	43	1281.80	2.87	0.96
5	26	790.26	4.65	0.93
7	19	587.58	6.26	0.89
9	15	471.73	7.79	0.87
11	12	385.25	9.54	0.87

Table IV - N = 127

Figure 2 contains a plot of the speedup versus number of machines, while Figure 3 displays the efficiency versus number of machines. As can readily be seen, for large problems the results are very encouraging. Indeed for N = 127 the algorithm remains over 85% efficient. This indicates that the message transfer time is successfully dominated by the time required for computations. However, for small problems the efficiency rapidly drops off, which indicates that this form of distributing the algorithm is not worthwhile for small problems.

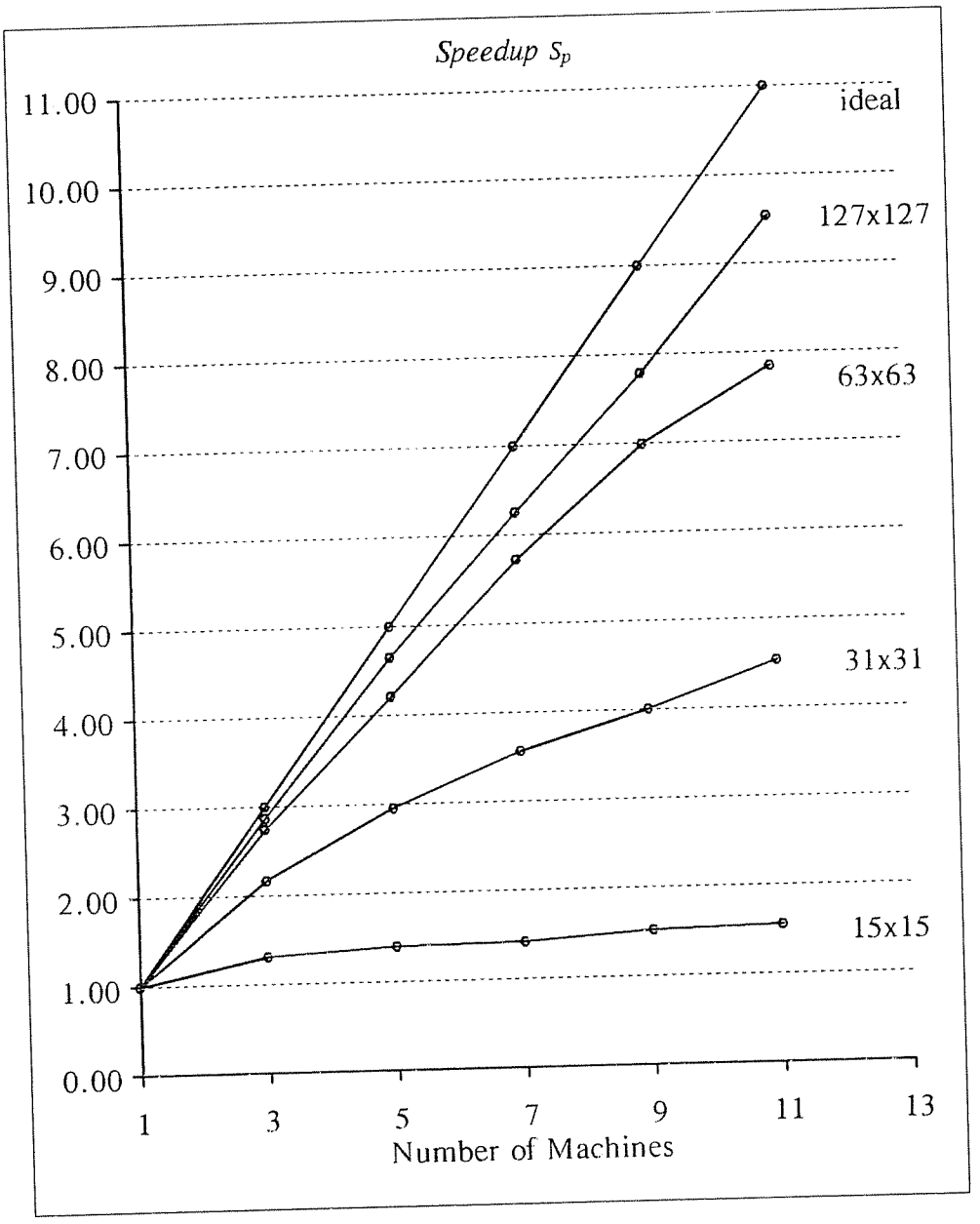

Speedup S_p
11.00
10.00
9.00
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
ideal
127x127
63x63
31x31
15x15
1
3
5
7
9
11
13
Number of Machines

Figure 2

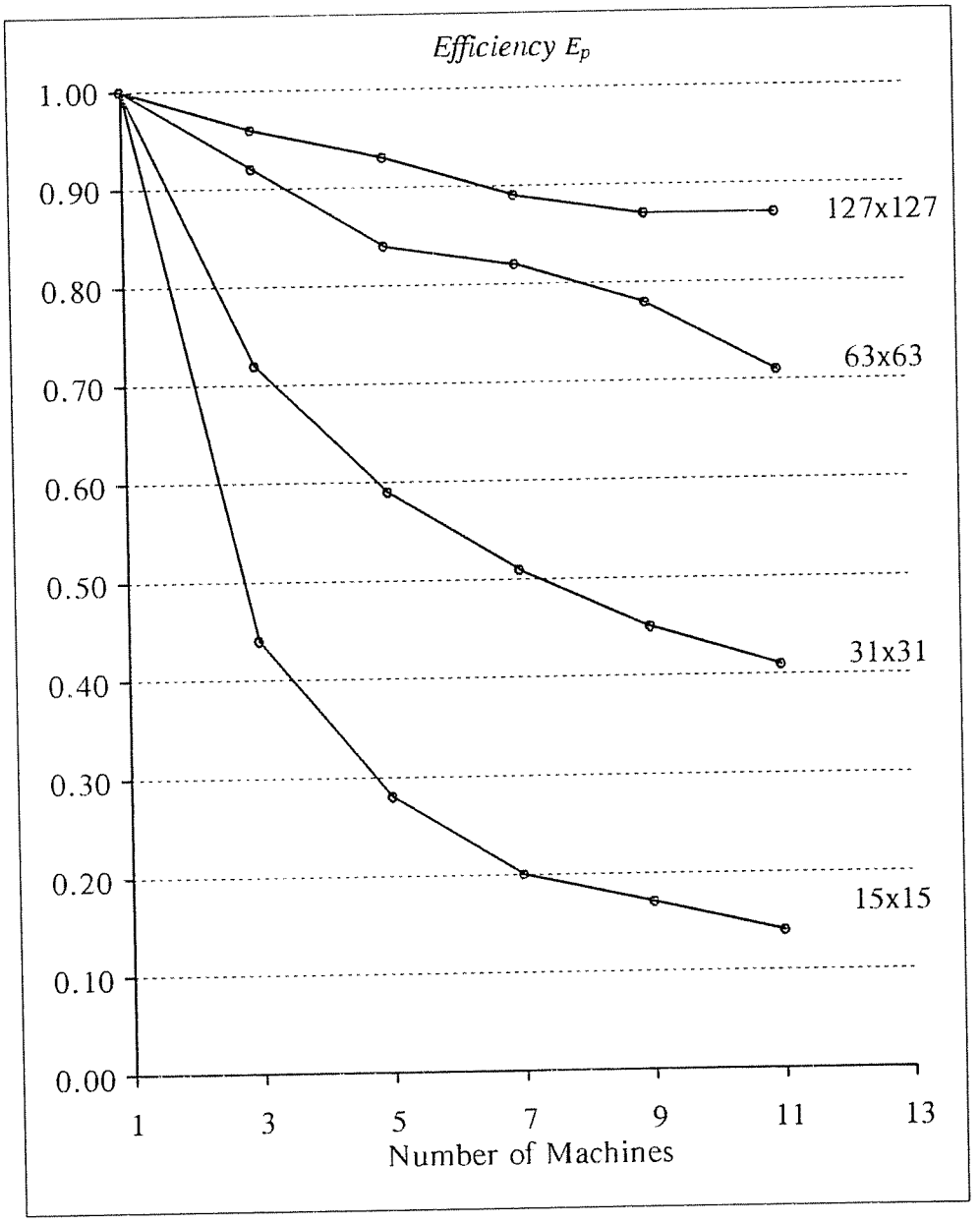

Efficiency E_p
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
127x127
63x63
31x31
15x15
1
3
5
7
9
11
13
Number of Machines

Figure 3

5. The Multigrid Method

The multigrid algorithm for the solution of (2.1) has been studied by many authors, see [8, 9, 10, 11]. Multigrid is a term used to describe an iterative technique which uses auxiliary grids which usually have significantly fewer points than the original grid. We will not attempt to describe all multigrid algorithms here, but rather will describe the one algorithm implemented on the Crystal multicomputer.

The algorithm chosen for implementation is known as the MGR[ν] algorithm. This algorithm was first described by Braess [12] (algorithm 2.1 in his paper) who analyzed the two grid version of what became known as MGR[0] for the Poisson equation in a general polygonal domain. Ries, Trottenberg and Winter [13] later analyzed the algorithm for the Poisson equation in a square for arbitrary ν. Their results agree with the result of Braess for the case $\nu = 0$. Kamowitz and Parter [14] extended the previous results for two grids for the MGR[0] case to the variable coefficient diffusion equation in a general polygonal domain. And finally Parter [15] extended the results of Ries Trottenberg and Winter and of Braess. He proved that the three grid rate of convergence in a general polygonal domain for MGR[0] for the variable coefficient diffusion equation is the same as the rate, $\rho \leq \frac{1}{2} + O(h)$, for the two grid algorithm.

5.1. The MGR[ν] Algorithm

In order to completely describe the MGR[ν] algorithm a number of spaces, operators and parameters need to be defined. In brief, each multigrid iteration consists of a small number of smoothing iterations, the transfer of the residual to a coarser grid, the solution of a related system of equations to compute the "coarse grid correction" and the updating of the smoothed values in the original, fine, grid by interpolating the coarse grid correction to the fine space. It should be noted that the multigrid algorithm itself can be used recursively to compute the coarse grid correction; this leads to a true multigrid algorithm. Also, additional smoothing steps can be done to the coarse grid correction while interpolating to finer grids.

First the general MGR[ν] multigrid algorithm will be presented, then the details concerning each stage of the algorithm will be described. In terms of the implementation on Crystal only a rudimentary understanding of the algorithm is necessary. However the details are important in terms of the actual performance on

Crystal.

The MGR[ν] algorithm uses t nested grids, where t is selected in advance of running the algorithm. The nested sequence of grids is labeled $\Omega_1 \supset \Omega_2 \supset \ldots \Omega_t$ where Ω_1 corresponds to Ω_h of section 2. Associated with each grid Ω_k is a positive definite, symmetric operator

$$L_k : \Omega_k \to \Omega_k.$$

To solve $L_1 U_1 = f_1$, where L_1 is the linear operator defined in equation (2.2) and f_1 is 0 for our particular problem the following algorithm is used.

Set $k := 1$, $U_k :=$ initial guess.

Algorithm MG(L_k, U_k, f_k, k);
(L_k given positive definite symmetric operator,
U_k given initial guess, returns value at next iteration,
f_k right hand side,
k grid layer)

(1) *Smooth:* Perform ν iterations of odd/even Gauss-Seidel relaxation on the problem $L_k U_k = f_k$ followed by one odd sweep. Store the results of this step in $\tilde{U}_k$.

(2) Compute the *Residual* r_k:

$$\text{Set } r_k := f_k - L_k \tilde{U}_k$$

Note that at the odd points $r_k = 0$.

(3) *Restrict* the residual r_k to Ω_{k+1}:

$$\text{Set } f_{k+1} := I_k^{k+1} r_k$$

(4) Consider computing the *Coarse grid correction:*

$$\text{Find } U_{k+1} \text{ such that } L_{k+1} U_{k+1} = f_{k+1}.$$

If $k+1 = t$ solve directly (i.e. return $U_t = L_t^{-1} f_t$).

Otherwise, set $U_{k+1} := 0$ and return MG($L_{k+1}, U_{k+1}, f_{k+1}, k+1$)

(5) *Interpolate* and update $\tilde{U}_k$:

$$\text{Set } U_k := \tilde{U}_k + I_{k+1}^k U_{k+1}.$$

(6) Return U_k, exit algorithm.

In the above description of algorithm MG(L_k, U_k, f_k, k) the details of the operators L_k, I_k^{k+1} and I_{k+1}^k were deliberately left out. Indeed, with the exception of Ω_1 which corresponds to Ω_h, the spaces Ω_k, $k = 2, 3, \cdots t$ have not yet been defined.

For clarity only the particulars for the two grid algorithm will be described fully. The details for the full t grid algorithm extend readily from the two grid description.

5.1.1. Additional Details of the MGR[v] Algorithm

Coarse Grid Spaces

Given N_1, recall the definition of Ω_1

$$\Omega_1 \equiv \Omega_h \equiv \left\{(x_i, y_j) \in \Omega \mid x_i = ih,\ y_j = jh,\ 1 \leq i, j \leq N_1\right\}.$$

Then the coarse grid spaces Ω_l, $l = 3, 5, \cdots$ correspond to setting

$$N_{new} := 2^{\frac{1-l}{2}}\left(N_1 + 1\right) - 1$$

and computing Ω_h with h now equal to $\frac{1}{N_{new}+1}$. The coarse grid spaces Ω_l, $l = 2, 4, \cdots$ correspond to the "black" points of Ω_{l-1}.

Communication between Spaces

The *interpolation* operator I_{k+1}^k is constructed as follows:

$$\left(I_{k-1}^k U\right)_{i,j} := U_{i,j} \text{ if the point } (x_i, y_j) \in \Omega_k$$

and for $(x_i, y_j) \in \Omega_k \setminus \Omega_{k+1}$ we require

$$\left[L_k\left(I_{k+1}^k U\right)\right]_{i,j} = 0. \tag{5.1}$$

Note that (5.1) results in an explicit equation for each point $(x_i, y_j) \in \Omega_k \setminus \Omega_{k+1}$.

For the *restriction* operator I_k^{k+1} we set

$$I_k^{k+1} := \frac{1}{2}\left(I_{k+1}^k\right)^T. \tag{5.2}$$

In step 3 of Algorithm MG applying the operator I_k^{k+1} to the residual reduces to dividing the residual by 2 on points of $\Omega_k \cap \Omega_{k+1}$

Coarse Grid Operators

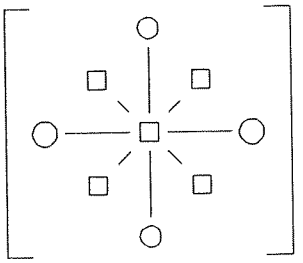
As is well known [11] the "ideal" choice for L_{k+1} is

$$\hat{L}_{k+1} := I_k^{k+1} L_k I_{k+1}^k.$$

With this choice of $\hat{L}_{k+1}$ for the coarse grid operator the two grid MGR[ν] algorithm converges in one step! However, $\hat{L}_{k+1}$ is a nine point operator as a straightforward calculation shows. In order to continue doing odd/even relaxation on each of the coarser grid layers we require $\hat{L}_{k+1}$ to be a five point operator.

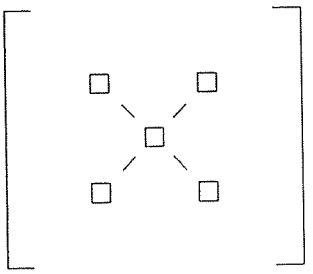
More specifically, the stencil for $\hat{L}_{k+1}$ (k odd) is of the form

$\hat{\mathbf{L}}_{\mathbf{k+1}} :=$



For the MGR[ν] algorithm we take L_{k+1} to be the "nearest" five points in the stencil for $\hat{L}_{k+1}$. E.g., the stencil for L_{k+1} (k odd) is

$\mathbf{L}_{\mathbf{k+1}} :=$



For k even, L_{k+1} corresponds to $L_{h'}$, where $h' = 2^{\frac{k}{2}} h$. It should be noted that this choice of L_{k+1} for even numbered grid layers results in the rotated grids characteristic of the MGR[ν] algorithm.

5.2. Implementing the MGR[ν] Algorithm on Crystal

The MGR[ν] algorithm of Section 5.1 was implemented on the Crystal multicomputer. Each grid Ω_k, $k = 1,2, \cdots, t$ is partitioned among p processors, just as for the red/black SOR algorithm of section 3. Steps 1-3 and 5 of algorithm MG are local steps, while step 4 is a global step. By a local step we mean a step of the algorithm where updating any particular point requires only the values of the four nearest neighboring points. The coarse solve on Ω_t, in step 4, is a global step since the values of the unknowns throughout all of Ω_t are required before the coarse grid correction can be computed. The purpose of this experimental study on the Crystal multicomputer is to determine whether there is any combination of number of grid layers and number of smoothing iterations (ν) for which the distributed portions of the algorithm effectively mask the deleterious effect of the global solution step.

As the number of grid layers increases, one hopes that the effect of the global solution step on the efficiency of the distributed algorithm should decrease. However, as we shall see, as the number of grid layers increases, the amount of work per grid layer decreases across all machines, and eventually there is a drop off in the efficiency of distributing the local steps across all the machines. For example, if the coarsest grid, Ω_t, has only one point on it, then the global solution step can be accomplished in one arithmetic operation. Unfortunately, the local operations on Ω_t are divided among the p processors in use, which means in this case that $p-1$ processors are idle. The tables and graphs following this section illustrate this effect.

The details of the implementation of the local steps (smoothing, computing the residual, restricting to coarser grids and interpolating to finer grids) are similar to the implementation of the red/black SOR algorithm. That is, send boundary data, update interior sections, receive boundaries from neighbors and then update the boundary values. These local steps are repeated for each of the t grid layers in use. Unfortunately as the number of grid layers increases, the number of points per grid layer decreases, and eventually the communication time for each step of the algorithm on these coarser grids dominates the computational time.

In addition to the local steps of the algorithm the coarse grid correction of step 4, on the coarsest grid, requires knowledge of the remaining unknowns in all of Ω_t. These unknowns are distributed among all p

processors in use. For this step each machine sends its unknowns to an additional, dedicated, node. This *master* node collects the unknowns from each of the *slave* nodes. These *slave* nodes correspond to the nodes used by the red/black SOR algorithm and perform the local operations. Once each *slave* node has sent its unknowns to this dedicated machine the coarse grid correction is computed using Gaussian elimination. The results of the coarse grid correction computed by the *master* node are then redistributed to each *slave* machine and the algorithm continues.

Of coarse a more easily parallelizable algorithm, such as red/black SOR could have been used to solve the coarse grid problem. This would have increased the observed efficiency of the experimental study. However, the cost would have been a slower running algorithm since for small sized problems (such as the coarse grid correction when using a fair number of grids) gaussian elimination is faster. The question of what technique to use to solve the coarse grid equation requires further study.

The coarse grid correction step results in the most serious bottleneck of the distributed version of the algorithm. During this step each node must remain idle while the coarse grid correction is computed. If the number of points in Ω_l is large (e.g. if $l = 2$ or 3 for example), then this step dominates the computational time of the algorithm. However, for the special case when the coarsest grid contains only one point, this transfer of unknowns to the *master* node is eliminated. For this special case the one node containing the coarse grid (with only one point in it) computes the coarse grid correction itself.

In addition to the bottleneck resulting from the coarse grid correction, there is another load balancing problem inherent in implementing the MGR[ν] algorithm on Crystal. As the grids get coarser eventually there are more machines in use than there are rows in the coarser grids. This results in the situation where some of the machines have no work to do and hence must sit idle for some portion of the algorithm. From a practical programming point of view this results in the added complication of keeping the machines synchronized throughout the iteration.

5.3. Experimental Results

The MGR[ν] algorithm as previously described was implemented and tested on the Crystal multicomputer. Tests were made with $N = 15$, 31, 63 and 127 and with ν equal to 0, 1 and 2. The number of

processors p used to perform steps 1-3 and 5 of algorithm MG was equal to 1, 3, 5, 7, 9 and 11. With the exception of the case $p=1$ and the case where Ω_l has one point in it one additional processor was used to compute the coarse grid correction. Thus, the total number of processors used when Ω_l had more than one point in it equaled 1, 4, 6, 8, 10 and 12. When Ω_l had only one point, the number of processors used equaled 1, 3, 5, 7, 9 and 11. Unfortunately due to physical constraints on the amount of memory in the node machines some combinations of the above parameters could not be tested and these cases are noted in the tables found in the appendix. Also, the single machine tests with $N=127$ were run on a lightly loaded VAX 750 (the same type of VAX as a node machine) running UNIX. By way of comparison a few runs were made with $N=63$ on both the VAX running UNIX and on a node machine. The times from both machines agreed to within a few seconds.

The purpose of our experimental study on the Crystal architecture is to determine what the optimal choice, if indeed there is one, of p and ν are for a particular size problem. To gain insight into this question it is worthwhile to look at both the observed rate of convergence and the distribution of computational work between the easily distributed steps of the algorithm and the coarse solve step.

The appendix contains the full set of observed CPU times and efficiencies for all the test problems. For expository simplicity only the case $N = 63$ will be discussed in this section. This case contains the full range of the parameters ν and number of grids and is representative of the other sized problems.

Figure 1 displays the observed rate of convergence for $N = 63$ and for $\nu = 0$, 1 and 2. Note that for 2 and 3 grids the observed rate of convergence is indeed bounded above by the predicted rate of

$$\frac{1}{2}\left(\frac{2\nu+1}{2\nu+1}\right)^{\frac{2\nu}{2\nu+1}}.$$

However, for $\nu = 1$ and 2 there is very little change in the rate of convergence as the number of grids increases. This is in some ways counter-intuitive and requires further theoretical investigation.

After observing the rate of convergence for each test case a crude count of the computational work of the algorithm was made. Since only a rough estimate of the work is of interest, 1 "work unit" was assigned to each unknown at each step of the algorithm. For example, with n points on a grid, n "work units" were counted during the smoothing step rather than $5n$ floating point operations which is formally more correct for

one iteration of Gauss-Seidel smoothing. The computational work required for convergence for each sized problem is proportional to the size of the problem.

Figure 2 displays a graph of the total computational work for $N = 63$ and figure 3 displays a graph of the ratio of work for the coarse solve step to the total computational work. Notice that the amount of work being done in the coarse solve step falls off rapidly.

Figures 4-6 display graphs of the observed efficiency for $\nu = 0$, 1 and 2. Each line corresponds to a different number of grid layers. The bottom line, representing the least efficient case displays the observed efficiency for 2 grids, while the bold line displays the observed efficiency for the special case where Ω_l contains only one point. Recall that in this case the *master* node does no work, so the number of processors used is simply the number used for steps 1-3 and 5 of the algorithm.

As the number of grids used increases, the efficiency coalesces. This is not surprising since beyond using a small number of grid layers there is not much difference in the amount of work being performed with respect to the number of grids used (see figure 2).

Unfortunately from a distributed algorithm point of view as the number of machines increases the efficiency drops off steadily. This particular implementation of the MGR[ν] algorithm is caught in the bind of either having too much work to do solving the coarse grid equations, or having too little work to do on the coarser grids while smoothing, computing the residual, etc. In addition, having to wait idly for the coarse grid equations to be computed is another limiting factor in terms of increasing the efficiency of the algorithm. In the special case where Ω_l has one point, this problem is somewhat alleviated, as can be seen in figures 3-5. Yet, as the number of machines increases, even in this case the amount of work remaining to be distributed among the processors is too small to effectively use them all efficiently.

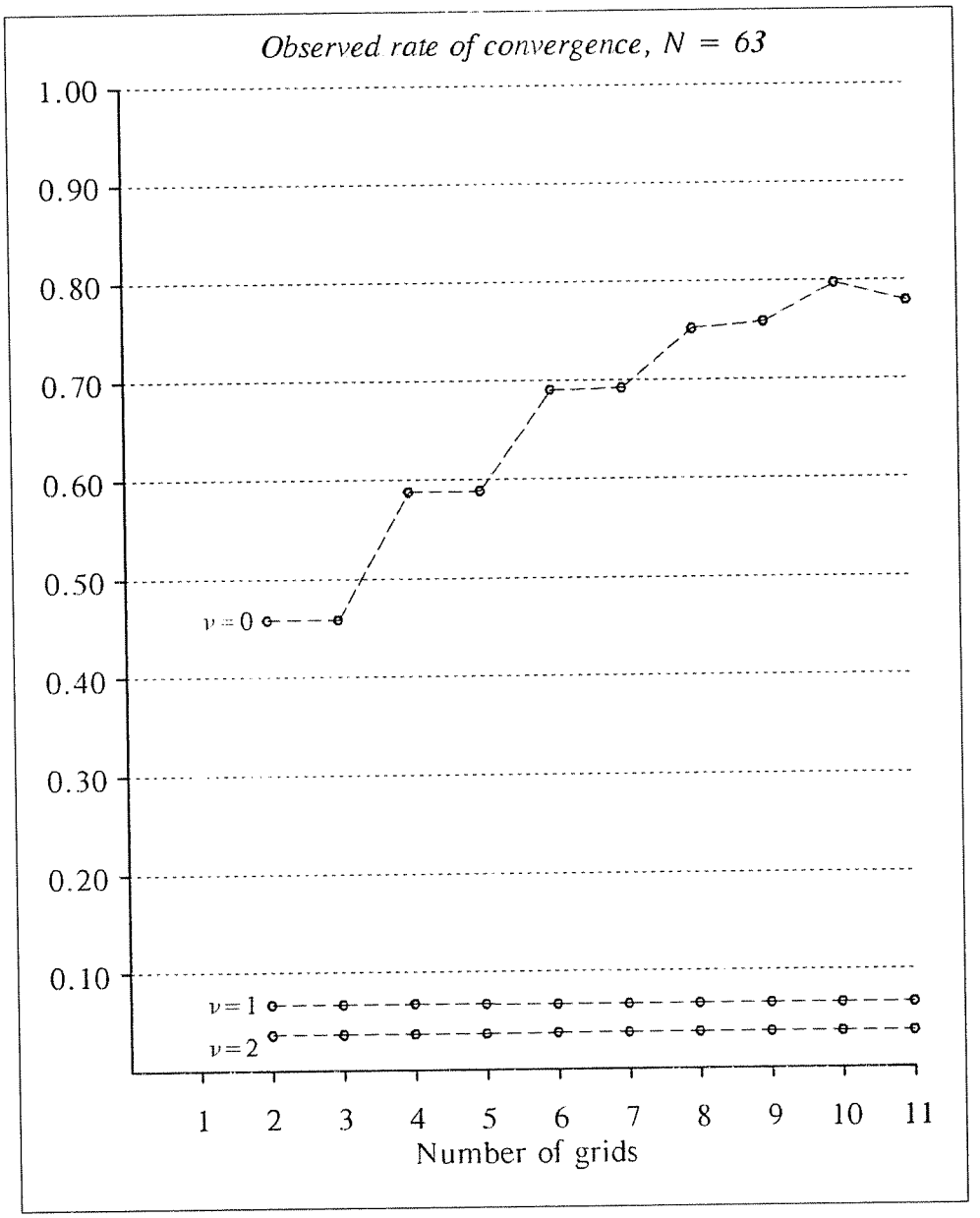

Observed rate of convergence, N = 63
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
ν = 0
ν = 1
ν = 2
1
2
3
4
5
6
7
8
9
10
11
Number of grids

Figure 1

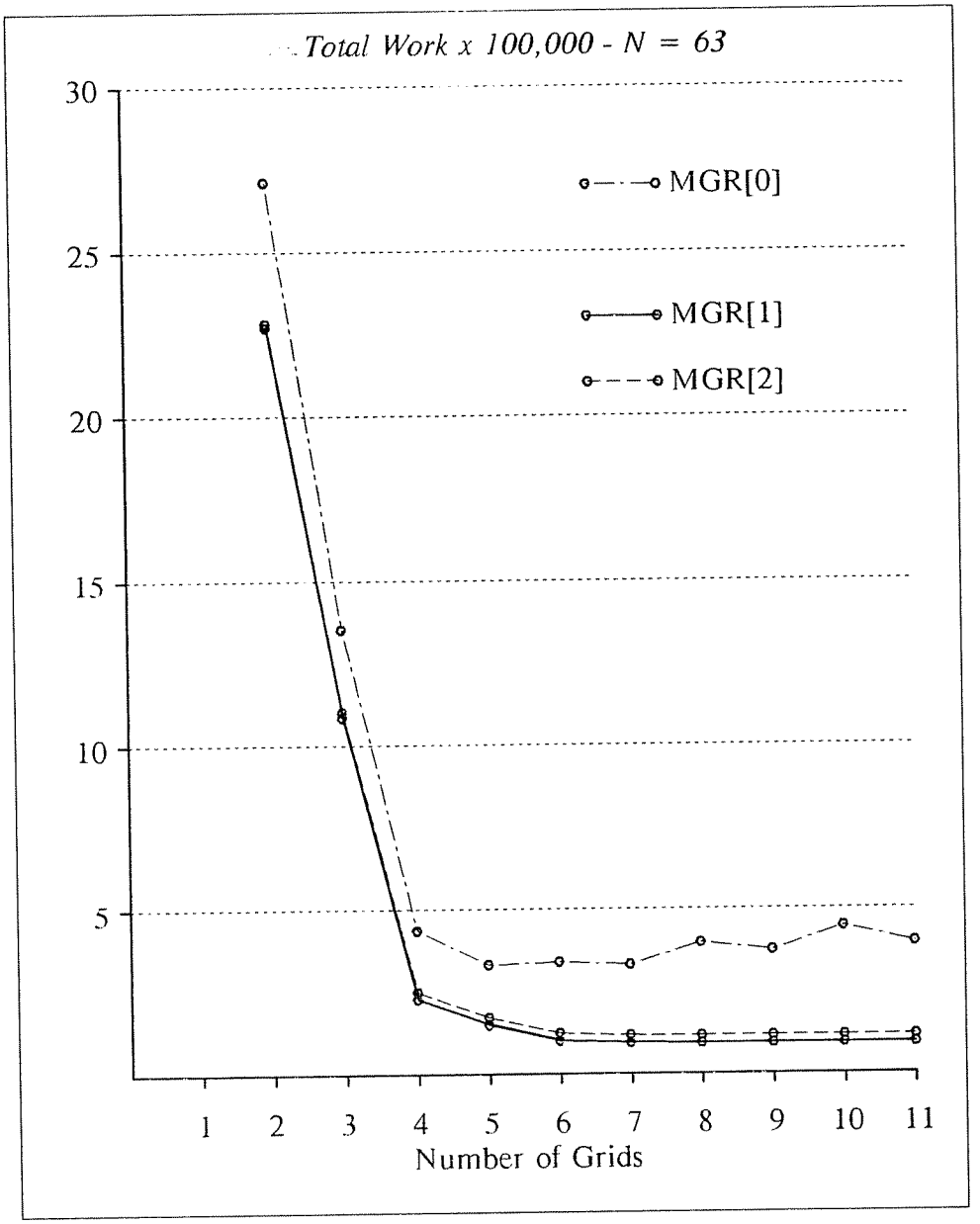

Total Work x 100,000 - N = 63
30
25
20
15
10
5
MGR[0]
MGR[1]
MGR[2]
1
2
3
4
5
6
7
8
9
10
11
Number of Grids

Figure 2

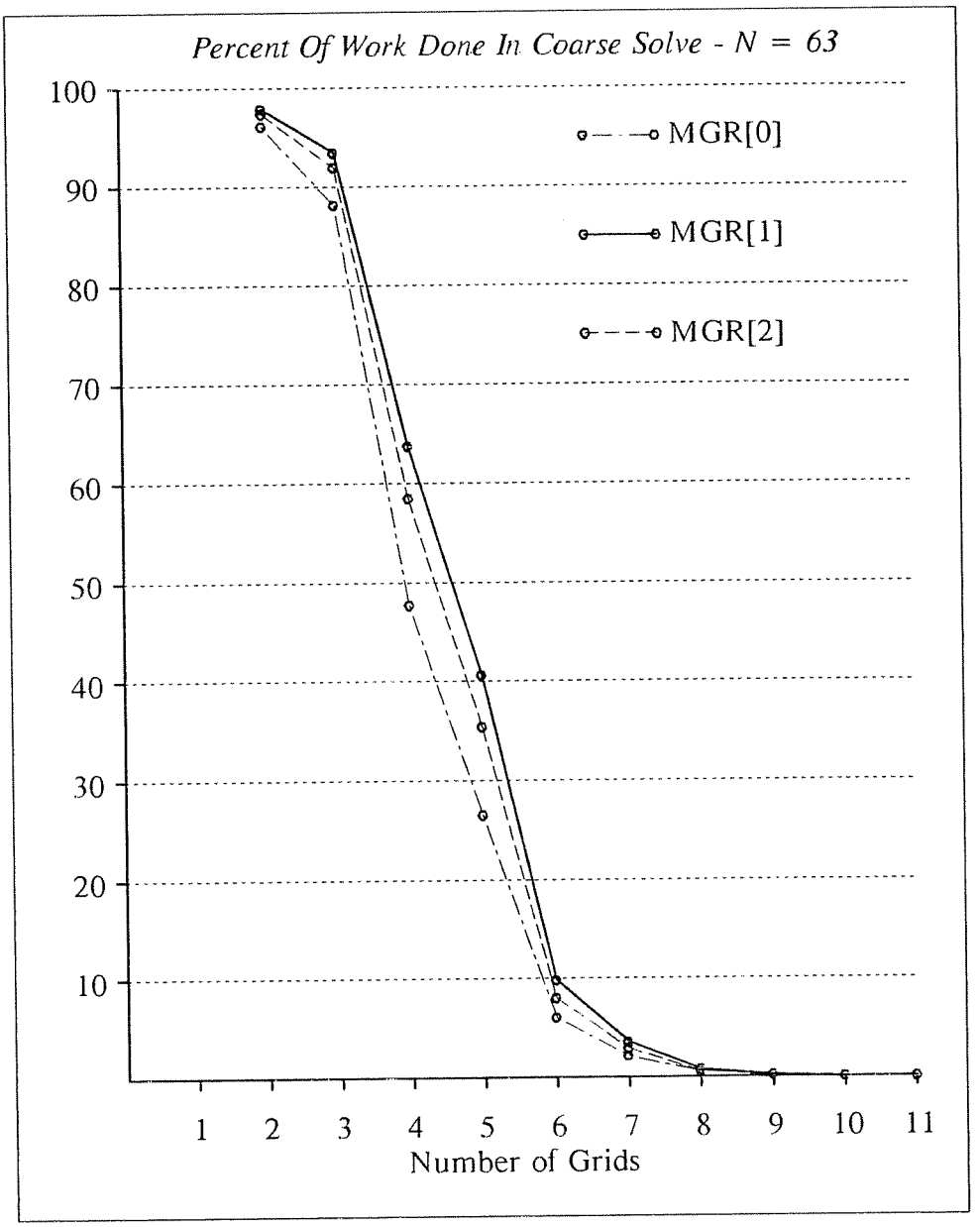

Percent Of Work Done In Coarse Solve - N = 63
100
90
80
70
60
50
40
30
20
10
MGR[0]
MGR[1]
MGR[2]
1
2
3
4
5
6
7
8
9
10
11
Number of Grids

Figure 3

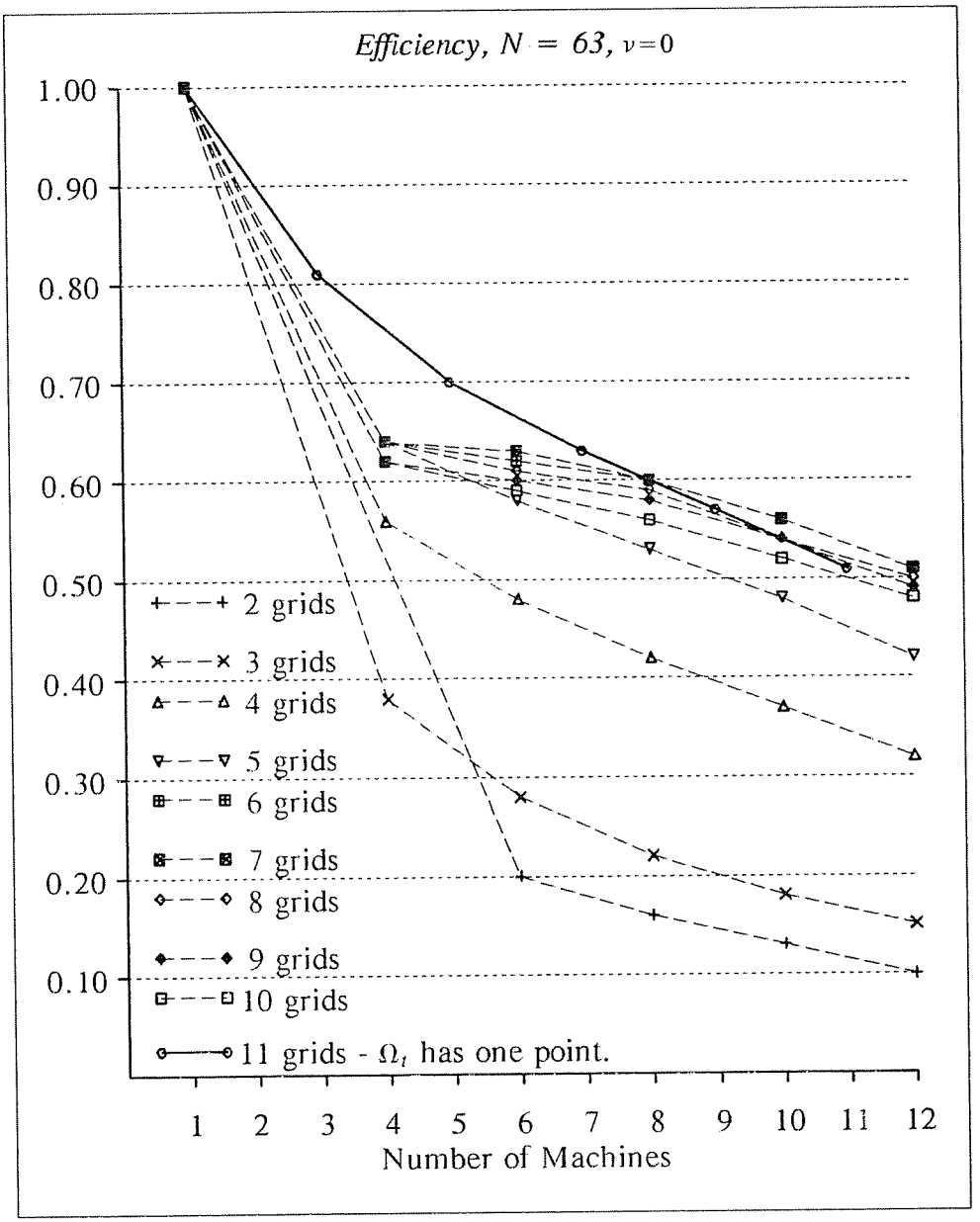

Efficiency, N = 63, ν=0
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
2 grids
3 grids
4 grids
5 grids
6 grids
7 grids
8 grids
9 grids
10 grids
11 grids - Ω1 has one point.
1 2 3 4 5 6 7 8 9 10 11 12
Number of Machines

Figure 4

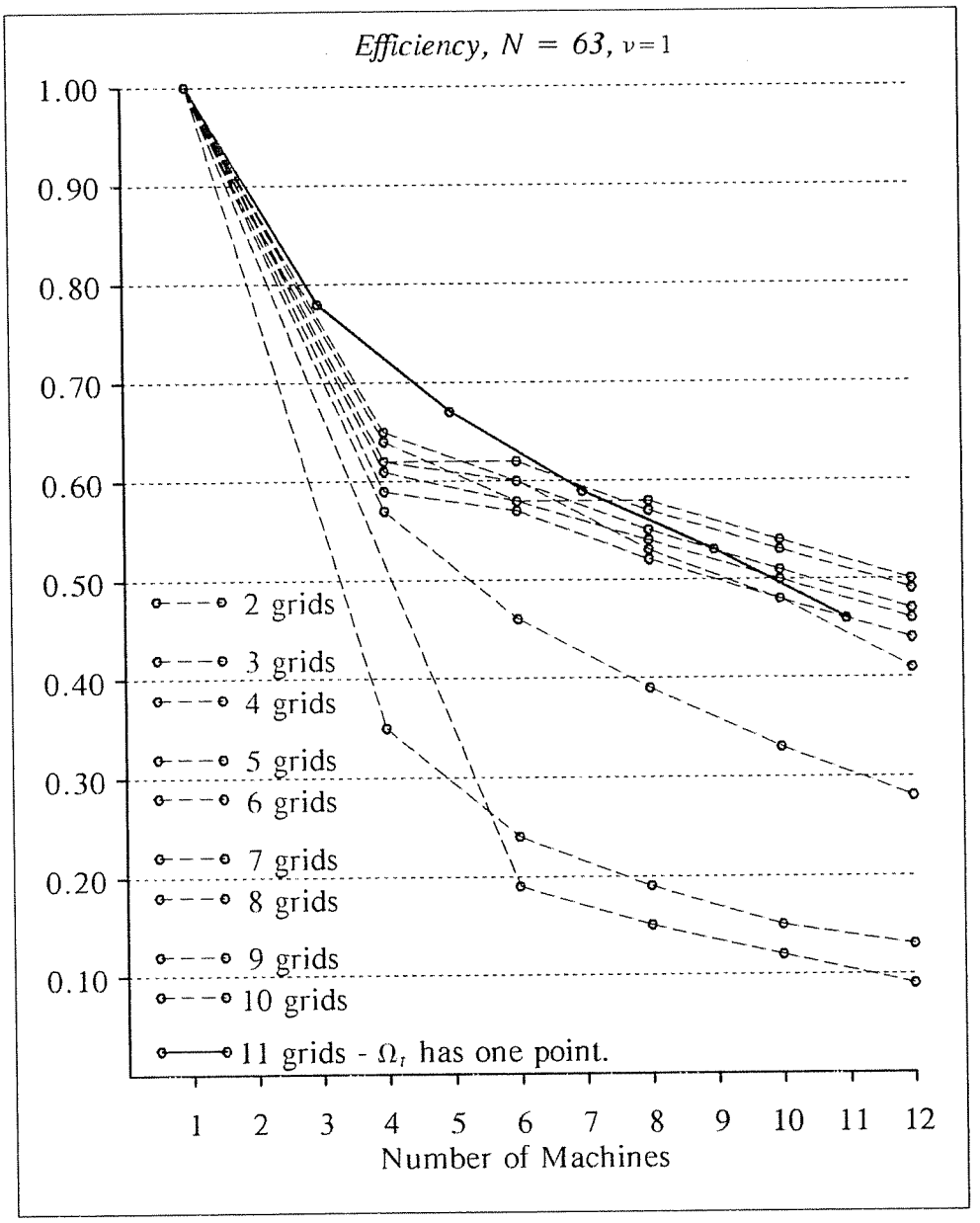

Efficiency, N = 63, ν=1
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
2 grids
3 grids
4 grids
5 grids
6 grids
7 grids
8 grids
9 grids
10 grids
11 grids - Ω_r has one point.
1 2 3 4 5 6 7 8 9 10 11 12
Number of Machines

Figure 5

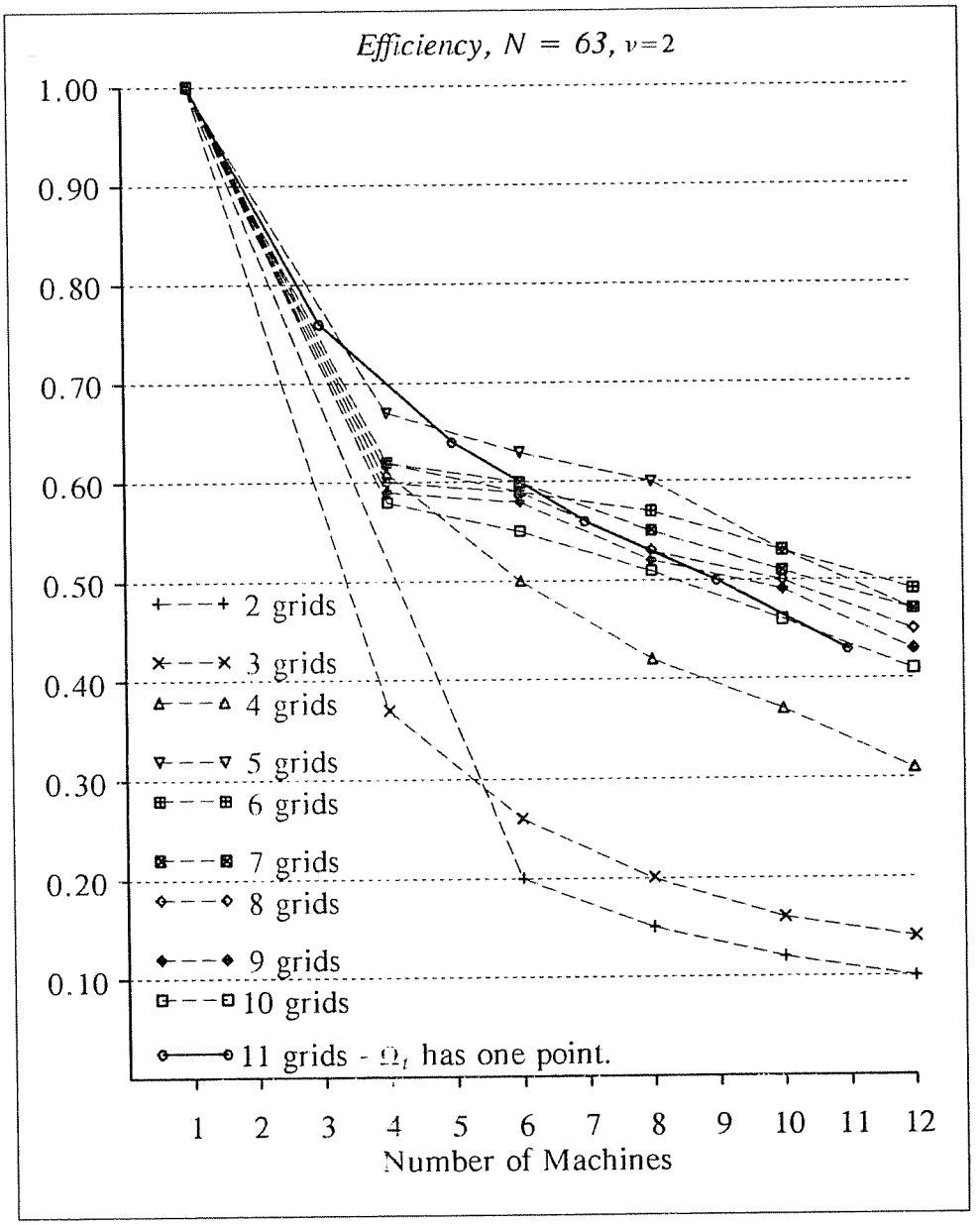

Efficiency, N = 63, ν=2
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
2 grids
3 grids
4 grids
5 grids
6 grids
7 grids
8 grids
9 grids
10 grids
11 grids - Ω_t has one point.
1 2 3 4 5 6 7 8 9 10 11 12
Number of Machines

Figure 6

6. Concluding remarks

Two approaches were presented for solving problem (2.2). The first approach, red/black SOR, was easy to implement on the Crystal architecture and the experimental results in terms of the observed efficiency for this algorithm were very encouraging.

The second approach, the MGR[ν] algorithm, was much more difficult to implement on the Crystal architecture. Alas, this particular implementation did not succeed in terms of high efficiency. Indeed this lack of high efficiency appears to be inherent in the algorithm itself.

There is one important saving grace in the MGR[ν] algorithm. While it might not lend itself to a distributed implementation, even the serial version is much faster than the red/black SOR algorithm. With $\nu = 1$, and the appropriate number of grids (depending upon N), the MGR[ν] algorithm was up to seventeen times faster than the serial version of the red/black SOR algorithm. In practical terms, this means that for a 100% efficient parallel implementation of the red/black SOR algorithm to be competitive with the MGR[ν] algorithm at least seventeen machines must be used for every one machine used for the MGR[ν] algorithm.

6.1. Suggestions for further work.

A number of questions for further research have been opened by this study. The first question is what happens if asynchronous smoothing is used? How much degradation in the rate of convergence of the red/black SOR and the MGR[ν] algorithms, if any, will there be? What kind of theoretical convergence results can be expected? This approach has been looked at by a number of people, see [16, 17]; however no clear answer has emerged.

Another question that would perhaps improve the efficiency of the MGR[ν] algorithm is: Is there some way to work on more than one grid layer at a time? For instance, perhaps by staggering the iterations among the grid levels each machine could work on a different grid layer, or perhaps even on more than one level. This idea has been investigated by Greenbaum [18]. Alternatively, is there some way for some of the idle machines to perform useful work while waiting for the solution of the coarse grid equations?

Finally, is it worthwhile to use a parallel technique for the solution on the coarsest grid? As stated earlier, this would increase the efficiency of the algorithm while perhaps resulting in some overall slowdown in the time required to converge, at least for the serial version.

Appendix

Tables 1-4 display the observed rate of convergence and the number of iterations required for each test problem. In tables 5-9, a-c can be found the observed CPU time for each test problem. 5a corresponds to $N = 15$, $\nu = 0$, 5b corresponds to $N = 15$, $\nu = 1$, etc. The efficiency for each test case is displayed in tables 9-12, a-c. Finally, table 13 contains the observed efficiency for the special case where Ω_I contains one point.

Observed rate of convergence / Number of iterations

Number of grids	$\nu=0$	$\nu=1$	$\nu=2$
2	.4628 / 10	.06154 / 3	.03228 / 3
3	.4639 / 10	.06221 / 3	.03277 / 3
4	.5964 / 13	.06396 / 4	.03357 / 3
5	.5984 / 12	.06133 / 4	.03402 / 3
6	.6633 / 15	.06133 / 4	.03409 / 3
7	.6474 / 13	.06086 / 4	.03412 / 3

Table 1 - N = 15

Observed rate of convergence / Number of iterations

Number of grids	$\nu=0$	$\nu=1$	$\nu=2$
2	.4585 / 9	.06582 / 3	.03546 / 3
3	.4591 / 9	.06583 / 3	.03564 / 3
4	.5909 / 12	.06667 / 3	.03641 / 3
5	.5931 / 12	.06585 / 3	.03645 / 3
6	.6950 / 15	.06513 / 3	.03653 / 3
7	.6971 / 15	.06483 / 3	.03685 / 3
8	.7457 / 18	.06483 / 3	.03685 / 3
9	.7308 / 16	.06493 / 3	.03688 / 3

Table 2 - N = 31

Observed rate of convergence / Number of iterations

Number of grids	$\nu=0$	$\nu=1$	$\nu=2$
2	.4583 / 9	.06711 / 3	.03656 / 3
3	.4586 / 9	.06708 / 3	.03660 / 3
4	.5879 / 11	.06791 / 3	.03741 / 3
5	.5889 / 11	.06757 / 3	.03743 / 3
6	.6912 / 14	.06661 / 3	.03750 / 3
7	.6920 / 14	.06649 / 3	.03757 / 3
8	.7619 / 17	.06647 / 3	.03762 / 3
9	.7590 / 16	.06648 / 3	.03769 / 3
10	.7980 / 19	.06648 / 3	.03769 / 3
11	.7798 / 17	.06651 / 3	.03770 / 3

Table 3 - N = 63

Observed rate of convergence / Number of iterations

Number of grids	$\nu=0$	$\nu=1$	$\nu=2$
2	na[1]	na	na
3	na	na	na
4	.5879 / 11	.06844 / 3	.03781 / 3
5	.5884 / 11	.06828 / 3	.03782 / 3
6	.6887 / 13	.06723 / 3	.03790 / 3
7	.6890 / 13	.06718 / 3	.03793 / 3
8	.7590 / 16	.06712 / 3	.03799 / 3
9	.7587 / 16	.06713 / 3	.03801 / 3
10	.8049 / 18	.06716 / 3	.03803 / 3
11	.7993 / 17	.06717 / 3	.03805 / 3
12	.8301 / 17	.06717 / 3	.03805 / 3
13	.8123 / 18	.06718 / 3	.03805 / 3

Table 4 - N = 127

[1]na means that this particular run could not be performed; usually due to size constraints.

Observed solution time - N = 15

$\nu = 0$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	6.37	5.08	4.63	4.75	4.78	4.82
3	7.09	5.99	4.55	4.52	4.65	4.32
4	9.41	7.26	6.57	6.54	6.48	6.30
5	9.15	7.79	6.59	6.61	6.42	6.22
6	11.64	10.76	9.58	9.55	9.26	9.14
7	10.29	9.88	8.85	8.78	8.51	8.47

Table 5a

Observed solution time - N = 15

$\nu = 1$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	2.69	2.10	1.97	2.03	2.07	2.10
3	2.89	2.07	1.88	1.87	1.89	1.79
4	3.85	3.19	2.89	2.87	2.87	2.81
5	4.05	3.75	2.90	3.24	3.20	3.25
6	4.13	4.26	3.85	3.80	3.79	3.72
7	4.21	4.60	4.19	4.14	4.11	4.10

Table 5b

Observed solution time - N = 15

$\nu = 2$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	3.00	2.26	2.16	2.19	2.21	2.24
3	3.43	2.55	2.31	2.27	2.26	2.20
4	3.54	3.15	2.81	2.76	2.76	2.71
5	3.74	3.66	2.81	3.20	3.19	3.15
6	3.82	4.19	3.84	3.79	3.78	3.72
7	3.89	4.59	4.22	4.22	4.16	4.14

Table 5c

Observed solution time - N = 31

$\nu = 0$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	34.3	25.8	24.0	23.1	22.8	22.5
3	30.9	18.1	14.4	14.2	13.6	13.2
4	36.5	18.1	14.0	12.4	11.6	10.7
5	37.4	18.7	14.2	12.5	11.5	10.4
6	46.9	24.4	18.7	16.4	15.2	13.8
7	47.5	25.4	19.5	17.1	15.7	14.4
8	57.2	31.9	25.0	22.2	20.3	19.0
9	51.1	29.0	22.8	20.2	18.6	17.4

Table 6a

Observed solution time - N = 31

$\nu = 1$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	17.8	13.9	18.2	12.9	12.7	12.7
3	14.8	8.6	7.6	7.1	6.9	6.8
4	12.4	6.1	4.7	4.2	3.9	3.6
5	12.6	6.6	4.8	4.4	4.1	3.7
6	12.6	6.9	5.0	4.8	4.5	4.1
7	12.8	7.4	5.4	5.1	4.7	4.4
8	12.8	7.6	6.1	5.5	5.1	4.8
9	12.9	7.9	6.4	5.7	5.3	5.1

Table 6b

Observed solution time - N = 31

$\nu = 2$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	19.1	14.3	13.4	13.1	12.9	12.9
3	17.0	9.4	8.2	7.6	7.3	7.3
4	15.0	7.6	5.8	5.1	4.7	4.4
5	15.4	8.1	5.8	5.6	5.1	4.8
6	15.5	8.7	6.4	6.1	5.6	5.3
7	15.7	9.2	6.9	6.6	6.1	5.7
8	15.8	9.9	7.9	7.2	6.7	6.3
9	15.9	10.2	8.4	7.5	7.1	6.7

Table 6c

Observed solution time - N = 63

$\nu = 0$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	259.3	na	211.9	208.2	206.3	207.8
3	182.8	122.2	111.9	106.5	103.8	103.4
4	147.3	66.5	51.6	44.2	40.3	38.7
5	144.1	56.2	42.0	34.2	30.1	28.2
6	178.6	69.7	48.0	37.6	32.1	29.2
7	179.7	70.3	48.0	37.8	32.3	29.4
8	218.1	86.5	59.5	46.8	40.1	36.3
9	206.0	82.7	57.1	44.7	38.2	35.2
10	244.6	99.5	69.3	54.8	47.0	43.0
11	219.3	89.9	62.7	49.7	42.8	39.2

Table 7a

Observed solution time - N = 63

$\nu = 1$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	162.5	na	141.0	139.7	139.0	140.9
3	102.6	74.0	69.9	67.8	66.8	67.4
4	56.9	25.1	20.6	18.3	17.1	16.8
5	53.8	20.8	15.0	12.7	11.3	10.8
6	51.5	20.3	15.0	11.1	9.5	8.7
7	51.7	20.7	14.0	11.3	9.7	8.9
8	51.7	21.1	14.4	11.7	10.1	9.2
9	51.8	21.4	14.7	12.0	10.4	9.5
10	51.9	21.8	15.3	12.5	10.8	9.9
11	52.0	22.2	15.6	12.6	11.0	10.3

Table 7b

Observed solution time - N = 63

$\nu = 2$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	167.7	na	142.0	140.5	140.0	141.4
3	111.5	76.3	71.4	69.0	67.9	68.3
4	67.1	27.7	22.4	19.8	18.4	17.9
5	68.0	25.4	17.9	14.3	12.8	12.2
6	63.0	25.3	17.8	13.9	11.9	10.9
7	63.4	25.9	17.6	14.3	12.4	11.3
8	63.5	26.4	18.0	14.9	12.9	11.9
9	63.7	27.0	18.5	15.3	13.2	12.3
10	63.8	27.5	19.3	15.8	13.8	12.8
11	63.9	27.9	19.9	16.3	14.2	13.5

Table 7c

Observed solution time - N = 127

$\nu = 0$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	na	na	na	na	na	na
3	na	na	na	na	na	na
4	781.0	na	324.4	296.8	281.9	271.3
5	697.3	na	214.7	185.3	169.2	158.2
6	732.1	na	164.5	129.4	109.3	95.8
7	730.4	na	156.7	119.7	98.7	84.3
8	890.8	na	188.7	143.5	117.1	99.2
9	890.2	na	188.8	143.8	117.4	99.1
10	1004.3	na	na	na	132.9	112.6
11	953.2	na	na	na	126.1	106.8
12	1059.3	na	na	na	na	121.2
13	1007.9	na	na	na	na	115.5

Table 8a

Observed solution time - N = 127

$\nu = 1$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	na	na	na	na	na	na
3	na	na	na	na	na	na
4	338.8	na	168.4	160.0	155.3	152.3
5	269.2	na	97.0	88.1	83.1	79.8
6	221.5	na	49.5	38.8	33.6	30.3
7	218.2	na	48.6	37.2	30.6	26.1
8	215.0	na	48.0	36.6	30.0	25.5
9	215.5	na	48.3	37.2	30.2	25.7
10	215.4	na	na	na	30.5	26.1
11	215.6	na	na	na	30.8	25.8
12	218.2	na	na	na	na	26.8
13	221.0	na	na	na	na	27.0

Table 8b

Observed solution time - N = 127

$\nu = 2$

Number of Grids	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
2	na	na	na	na	na	na
3	na	na	na	na	na	na
4	380.2	na	174.7	165.2	159.2	155.7
5	318.1	na	104.2	93.5	87.4	83.4
6	270.2	na	60.4	46.7	38.6	34.0
7	268.3	na	59.9	45.7	37.5	32.3
8	266.1	na	59.6	45.2	37.2	31.9
9	267.1	na	60.1	45.8	37.9	32.2
10	267.6	na	na	na	38.1	32.7
11	268.2	na	na	na	38.6	33.2
12	270.0	na	na	na	na	33.1
13	270.3	na	na	na	na	34.1

Table 8c

E_p - N = 15
$\nu = 0$

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	1.0	.32	.23	.17	.14	.11
3	1.0	.29	.26	.19	.15	.14
4	1.0	.32	.24	.18	.14	.13
5	1.0	.29	.23	.18	.14	.12
6	1.0	.27	.20	.15	.13	.11

Table 9a

E_p - N = 15
$\nu = 1$

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	1.0	.32	.23	.17	.13	.11
3	1.0	.35	.26	.19.15	.14	
4	1.0	.30	.23	.17	.14	.11
5	1.0	.27	.23	.16	.13	.10
6	1.0	.24	.18	.14	.11	.09

Table 9b

E_p - N = 15
$\nu = 2$

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	1.0	.33	.23	.18	.14	.11
3	1.0	.34	.25	.19	.15	.13
4	1.0	.28	.21	.16	.13	.11
5	1.0	.26	.23	.15	.12	.10
6	1.0	.23	.17	.12	.10	.08

Table 9c

E_p - N = 31
$\nu = 0$

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	1.0	.33	.24	.18	.15	.13
3	1.0	.43	.36	.27	.23	.19
4	1.0	.50	.43	.37	.32	.28
5	1.0	.50	.44	.38	.37	.30
6	1.0	.48	.42	.36	.31	.28
7	1.0	.47	.41	.35	.31	.28
8	1.0	.45	.38	.32	.28	.25

Table 10a

E_p - N = 31
$\nu = 1$

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	1.0	.32	.17	.18	.14	.12
3	1.0	.43	.33	.26	.22	.18
4	1.0	.51	.43	.37	.32	.28
5	1.0	.48	.44	.36	.31	.28
6	1.0	.46	.42	.33	.28	.26
7	1.0	.44	.39	.32	.27	.24
8	1.0	.42	.35	.29	.25	.22

Table 10b

E_p - N = 31
$\nu = 2$

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	1.0	.34	.24	.18	.14	.12
3	1.0	.45	.34	.28	.23	.19
4	1.0	.50	.43	.37	.32	.28
5	1.0	.47	.44	.34	.31	.27
6	1.0	.44	.40	.32	.28	.25
7	1.0	.43	.38	.30	.26	.23
8	1.0	.40	.33	.27	.23	.21

Table 10c

E_p - N = 63
$\nu = 0$

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	1.0	na	.20	.16	.13	.10
3	1.0	.38	.28	.22	.18	.15
4	1.0	.56	.48	.42	.37	.32
5	1.0	.64	.58	.53	.48	.42
6	1.0	.64	.62	.60	.56	.51
7	1.0	.64	.63	.60	.56	.51
8	1.0	.64	.61	.59	.54	.50
9	1.0	.62	.60	.58	.54	.49
10	1.0	.62	.59	.56	.52	.48

Table 11a

E_p - N = 63
$\nu = 1$

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	1.0	na	.19	.15	.12	.09
3	1.0	.35	.24	.19	.15	.13
4	1.0	.57	.46	.39	.33	.28
5	1.0	.65	.60	.53	.48	.41
6	1.0	.64	.58	.58	.54	.50
7	1.0	.62	.62	.57	.53	.49
8	1.0	.62	.60	.55	.51	.47
9	1.0	.61	.58	.54	.50	.46
10	1.0	.59	.57	.52	.48	.44

Table 11b

E_p - N = 63
ν=2

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	1.0	na	.20	.15	.12	.10
3	1.0	.37	.26	.20	.16	.14
4	1.0	.61	.50	.42	.37	.31
5	1.0	.67	.63	.60	.53	.49
6	1.0	.62	.59	.57	.53	.49
7	1.0	.62	.60	.55	.51	.47
8	1.0	.60	.59	.53	.50	.45
9	1.0	.59	.58	.52	.49	.43
10	1.0	.58	.55	.51	.46	.41

Table 11c

E_p - N = 127
ν=0

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	na	na	na	na	na	na
3	na	na	na	na	na	na
4	1.0	na	.40	.33	.28	.24
5	1.0	na	.54	.47	.41	.37
6	1.0	na	.74	.71	.67	.63
7	1.0	na	.78	.76	.74	.72
8	1.0	na	.78	.78	.77	.75
9	1.0	na	.78	.77	.76	.75
10	1.0	na	na	na	.76	.74
11	1.0	na	na	na	.76	.74
12	1.0	na	na	na	na	.74

Table 12a

E_p - N = 127
$\nu = 1$

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	na	na	na	na	na	na
3	na	na	na	na	na	na
4	1.0	na	.33	.26	.22	.18
5	1.0	na	.47	.39	.32	.28
6	1.0	na	.74	.71	.66	.61
7	1.0	na	.75	.74	.71	.70
8	1.0	na	.75	.74	.72	.71
9	1.0	na	.74	.73	.71	.70
10	1.0	na	na	na	.70	.69
11	1.0	na	na	na	.70	.70
12	1.0	na	na	na	na	.68

Table 12b

E_p - N = 127
$\nu = 2$

Number of Grids	1 machine	4 machines	6 machines	8 machines	10 machines	12 machines
2	na	na	na	na	na	na
3	na	na	na	na	na	na
4	1.0	na	.37	.29	.24	.20
5	1.0	na	.51	.43	.36	.32
6	1.0	na	.74	.73	.70	.66
7	1.0	na	.75	.74	.71	.70
8	1.0	na	.74	.74	.71	.70
9	1.0	na	.74	.73	.70	.69
10	1.0	na	na	na	.70	.68
11	1.0	na	na	na	.69	.67
12	1.0	na	na	na	na	.67

Table 12c

Case Ω_t has one point.

ν	1 machine	3 machines	5 machines	7 machines	9 machines	11 machines
N = 15, 7 grids.						
0	1.0	.35	.23	.17	.13	.11
1	1.0	.31	.20	.15	.11	.09
2	1.0	.28	.18	.13	.10	.09
N = 31, 9 grids.						
0	1.0	.59	.45	.36	.31	.27
1	1.0	.54	.40	.32	.27	.23
2	1.0	.52	.38	.30	.25	.22
N = 63, 11 grids.						
0	1.0	.81	.70	.63	.57	.51
1	1.0	.78	.67	.59	.53	.46
2	1.0	.76	.64	.56	.50	.43
N = 127, 13 grids.						
0	1.0	na	na	na	na	.79
1	1.0	na	na	na	na	.74
2	1.0	na	na	na	na	.72

Table 13

References

1. R. Finkel, M. Solomon, D. DeWitt, and L. Landweber, "The Charlotte Distributed Operating System: Part IV of the first report on the Crystal project," Technical Report 502, University of Wisconsin-Madison Computer Sciences (July 1983).

2. D. DeWitt, R. Finkel, and M. Solomon, "The CRYSTAL Multicomputer: Design and Implementation Experience," Technical Report 553, University of Wisconsin-Madison Computer Sciences (September 1984).

3. D. P. O'Leary and G. W. Stewart, "Data-Flow Algorithms for Parallel Matrix Computtions," *Communications of the ACM* **28**(8) pp. 840-853 (August 1985).

4. D. Young, *Iterative Solution of Large Linear Systems*, Academic Press, New York (1971).

5. L. A. Hageman and D. Young, *Applied Iterative Methods*, Academic Press, New York (1981).

6. R. S. Varga, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey (1962).

7. B. L. Buzbee, D. Boley, and S. V. Parter, "Applications of Block Relaxation," Proceedings of the Fifth Symposium on Reservoir Simulation, Society of Petroleum Engineers of AIME, Denver, Colorado (Feb 1-2, 1979).

8. A. Brandt, "Multilevel Adaptive Solutions to Boundary-value Problems," *Math. Comp.* **31** pp. 333-390 (1977).

9. J. E. Dendy, Jr., "Black Box Multigrid," *J. Comput. Phys.* **48** pp. 366-386 (1982).

10. W. Hackbusch, "Convergence of Multi-grid Iterations Applied to Difference Equations," *Math. Comp.* **34** pp. 425-440 (1980).

11. S. McCormick and J. Ruge, "Multigrid Methods for Variational Problems," *SIAM Journal of Numerical Analysis* **19** pp. 924-929 (1982).

12. D. Braess, "The Contraction Number of a Multigrid Method for Solving the Poisson Equation," *Numer. Math.* **37** pp. 387-404 (1981).

13. M. Ries, U. Trottenberg, and G. Winter, "A Note on MGR Methods," *Linear Algebra Appl.* **49** pp. 1-26 (1983).

14. D. Kamowitz and S. V. Parter, "On MGR[ν] Multigrid Methods," Technical Report 575, University of Wisconsin-Madison Computer Sciences (January 1985).

15. S. V. Parter, "On an Estimate for the Three-Grid MGR Multigrid Method," Technical Report 610, University of Wisconsin-Madison Computer Sciences (August 1985).

16. D. Chazan and W. Miranker, "Chaotic Relaxation," *Linear Algebra Appl.* **2** pp. 199-222 (1969).

17. W. Miranker, "Parallel Methods for Solving Equations," *Mathematics and Computers in Simulation* **XX** pp. 93-101 (1978).

18. A. Greenbaum, "A Multigrid Method for Multiprocessors," Technical Report UCRL 92211, Lawrence Livermore National Laboratory (February 12, 1985).